

50A 11/31/01
IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re U.S. Patent Application of)
)
SHIMIZU et al.)
)
Application Number: To be assigned)
)
Filed: Concurrently herewith)
)
For: SIMD OPERATION SYSTEM CAPABLE OF)
DESIGNATING PLURAL REGISTERS)



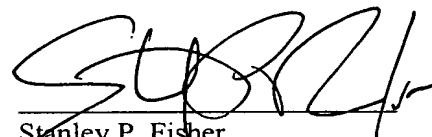
Honorable Assistant Commissioner
for Patents
Washington, D.C. 20231

**NOTICE OF PRIORITY
UNDER 35 U.S.C. 119
AND THE INTERNATIONAL CONVENTION**

Sir:

In the matter of the above-captioned application for a United States patent, notice is hereby given that the Applicant claims the priority date of November 8, 2000, the filing date of the corresponding Japanese patent priority application 2000-340239.

A certified copy of corresponding Japanese patent application 2000-340239 is being submitted herewith. The Examiner is most respectfully requested to acknowledge receipt of the certified copy in due course.


Stanley P. Fisher
Registration Number 24,344

REED SMITH HAZEL & THOMAS LLP
3110 Fairview Park Drive
Suite 1400
Falls Church, Virginia 22042
(703) 641-4200

JUAN CARLOS A. MARQUEZ
Registration No. 34,072

September 4, 2001

**PATENT OFFICE
JAPANESE GOVERNMENT**



This is to certify that the annexed is a true copy of the following application as filed with this office.

Date of Application : November 8, 2001
Application Number : Patent Application No. 2000-340239
Applicant (s) : Hitachi, Ltd.

Dated this 10th day of August, 2001

Kouzou OIKAWA
Commissioner,
Patent Office

Certificate No. 2001-3072202

日 本 国 特 許 庁
JAPAN PATENT OFFICE



別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application: 2000年11月 8日

出 願 番 号

Application Number: 特願2000-340239

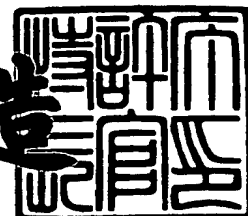
出 願 人

Applicant(s): 株式会社日立製作所

2001年 8月10日

特 許 庁 長 官
Commissioner,
Japan Patent Office

及 川 耕 造



出証番号 出証特2001-3072202

【書類名】 特許願

【整理番号】 NT00P0832

【提出日】 平成12年11月 8日

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 15/80
G06F 15/82

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目 2 8 0 番地 株式会社日立製作所 中央研究所内

【氏名】 清水 健央

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目 2 8 0 番地 株式会社日立製作所 中央研究所内

【氏名】 荒川 文男

【特許出願人】

【識別番号】 000005108

【氏名又は名称】 株式会社日立製作所

【代理人】

【識別番号】 100068504

【弁理士】

【氏名又は名称】 小川 勝男

【電話番号】 03-3661-0071

【選任した代理人】

【識別番号】 100086656

【弁理士】

【氏名又は名称】 田中 恭助

【電話番号】 03-3661-0071

【選任した代理人】

【識別番号】 100094352

【弁理士】

【氏名又は名称】 佐々木 孝

【電話番号】 03-3661-0071

【手数料の表示】

【予納台帳番号】 081423

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【ブルーフの要否】 要

【書類名】 明細書

【発明の名称】 複数レジスタ指定が可能な SIMD 演算方式

【特許請求の範囲】

【請求項 1】

命令コードと、少なくとも 1 つのレジスタ指定フィールドからなる演算命令を有し、上記少なくとも 1 つのレジスタ指定フィールドは連続番号の複数個のレジスタが指定可能であることを特徴とするプロセッサ。

【請求項 2】

任意数のレジスタ指定フィールドの内、1 フィールドで複数のリードレジスタを指定するデコーダと、該デコーダからの出力に従って連続番号の複数個のレジスタ内データを出力するレジスタファイルとを具備することを特徴とするプロセッサ。

【請求項 3】

任意数のレジスタ指定フィールドの内、1 フィールドで複数のライトレジスタを指定するデコーダと、該デコーダからの出力に従って連続番号の複数個のレジスタに値を書き込めるレジスタファイルとを具備することを特徴とするプロセッサ。

【請求項 4】

請求項 2 もしくは 3 記載のプロセッサにおいて、上記レジスタファイルは複数のバンクを有し、前記複数のバンクからリードまたはライトを行うことにより、各バンクのリードまたはライトポート数をレジスタ指定フィールド数以下に制限して、フィールド数より多くのリードまたはライトを行うことによる回路規模の増大を抑制したプロセッサ。

【請求項 5】

請求項 1、2 又は 3 のいずれかに記載のプロセッサにおいて、該連続番号の複数個のレジスタ数は 2 の n 乗個（ n は自然数）と限定して、レジスタ選択回路の削減を可能としたプロセッサ。

【請求項 6】

請求項 1 から 5 のいずれかに記載のプロセッサにおいて、さらにリードレジス

タ指定フィールド数より多くの数のレジスタからデータをリードするため、ライトレジスタへの書き込みデータ数より多くのリードレジスタからの読み出しデータ数に対応可能なデータパック演算を、ライトレジスタに無効な部分を生成することなく実現可能であることを特徴とするプロセッサ。

【請求項 7】

請求項 1 から 5 のいずれかに記載のプロセッサにおいて、さらにライトレジスタ指定フィールド数より多い数のレジスタにライトできることから、リードレジスタの読み出しデータ数より多くのライトレジスタへの書き込みデータ数に対応可能なデータアンパック演算を、ライトを複数回に分けることなく並列的に実現可能であることを特徴とするプロセッサ。

【請求項 8】

請求項 1 から 5 のいずれかに記載のプロセッサにおいて、さらにライトレジスタ指定フィールド数より多くのレジスタにライトできることから、入力データ幅より広いデータ幅の出力を行う演算を、入力データに無効部分を作ることなく、さらにデータ幅の広い特別なレジスタを実装することなく、実現可能であることを特徴とするプロセッサ。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本発明は、S I M D (Single Instruction Multiple Data) プロセッサにおけるレジスタ指定方式および、レジスタ内データ整列処理に関するものであり、S I M D の並列処理性能を低下させる惧れの有るレジスタ内データ整列を高速に行う手段に関するものである。

【0 0 0 2】

また D S P (Digital Signal Processing) 命令として標準的な積和演算に関して、S I M D の並列性を維持したまま、精度を落とすことなく演算させることが可能な手段に関するものである。

【0 0 0 3】

【従来の技術】

3次元グラフィックス等で必要となるベクトル演算処理を行う際に、1つのレジスタ指定フィールドで連続した複数のレジスタを指定できるようにして、ベクトル演算を行わせるような方法が、特開平10-124484の特許で既に公開されている。

【0004】

また、レジスタ内データ整列命令としては、モトローラ社が開発したマルチメディア向け命令セットAltiVecの、「AltiVec Programming Interface Manual」に4オペランドまで指定可能な各種データ整列命令が記述されている。

【0005】

さらに積和演算に関しては、SIMDの並列性を4並列から2並列に半減させた形で実現するような精度落ちのない積和演算命令が、日立製作所とSTマイクロエレクトロニクスの共同開発のSH5アーキテクチャで定義されている。

【0006】

【発明が解決しようとする課題】

しかし、特開平10-124484に示されたベクトル演算処理においては、複数のレジスタ指定において、4の倍数の番号のみしか指定できない構成と成っているため自由度が乏しく、またAltiVecのデータ整列命令においては、演算装置が大型で高価である上にソースレジスタとして3個しか指定できず、データパックやアンパックといったSIMD特有の演算を効率良く行うことができない。従って、SIMDの並列性を十分に達成できていない。

【0007】

本発明の主たる目的は、SIMD命令の効果を最大限にあげるための、データ整列に関する手段を提供するものである。

【0008】

また本発明の他の目的は、DSP命令として標準的な積和演算命令をSIMDの並列性を落とすことなく、しかも精度を保ったまま実現する1手段を提供するものである。

【0009】

【課題を解決するための手段】

本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

【0010】

本発明は、命令コードと、少なくとも1つのレジスタ指定フィールドからなる演算命令を有し、少なくとも1つのレジスタ指定フィールドは連続番号の複数個のレジスタが指定可能であることを特徴とする。

【0011】

さらに本発明は、任意数のレジスタ指定フィールドの内、1フィールドで複数のリードレジスタを指定するデコーダと、デコーダからの出力に従って連続番号の複数個のレジスタ内データを出力するレジスタファイルとを具備することを特徴とする。

【0012】

さらに本発明は、任意数のレジスタ指定フィールドの内、1フィールドで複数のライトレジスタを指定するデコーダと、デコーダからの出力に従って連続番号の複数個のレジスタに値を書き込めるレジスタファイルとを具備することを特徴とする。

【0013】

上記レジスタファイルは複数のバンクを有し、複数のバンクからリードまたはライトを行うことにより、各バンクのリードまたはライトポート数をレジスタ指定フィールド数以下に制限して、フィールド数より多くのリードまたはライトを行うことによる回路規模の増大を抑制したものである。

【0014】

上記連続番号の複数個のレジスタ数は2のn乗個（nは自然数）と限定して、レジスタ選択回路の削減を可能としたものである。

【0015】

さらにリードレジスタ指定フィールド数より多くの数のレジスタからデータをリードするため、ライトレジスタへの書き込みデータ数より多くのリードレジスタからの読み出しデータ数に対応可能なデータパック演算を、ライトレジスタに

無効な部分を生成することなく実現可能であることを特徴とする。

【0016】

又さらにライトレジスタ指定フィールド数より多い数のレジスタにライトできることから、リードレジスタの読み出しデータ数より多くのライトレジスタへの書き込みデータ数に対応可能なデータアンパック演算を、ライトを複数回に分けることなく並列的に実現可能であることを特徴とする。

【0017】

さらにライトレジスタ指定フィールド数より多くのレジスタにライトできることから、入力データ幅より広いデータ幅の出力を行う演算を、入力データに無効部分を作ることなく、さらにデータ幅の広い特別なレジスタを実装することもなく、実現可能であることを特徴とする。

【0018】

さらにデータ整列を行うパイプなど、レジスタ指定フィールドの数以上のデータを必要とするパイプに、それぞれデータを供給するために、レジスタと演算パイプ間には、汎用のバスの他に、複数本のデータ用バスを設けていることを特徴とする。

【0019】

さらにデータアンパック命令、並べ替え命令、行列演算命令、積和演算命令など、複数個の結果を出力するパイプのために、レジスタと演算パイプ間に、レジスタ書き込み用として複数本のデータ用バスを設けていることを特徴とする。

【0020】

【発明の実施の形態】

以下、図面を参照して本発明の実施例を詳細に説明する。尚、実施例を説明するための全図において、同一機能を有するものは同一符号を付け、その繰り返しの説明は省略する。

【0021】

図1には、本発明の実施例であるSIMDプロセッサのCPUブロック図の概略を示す。ただし、このCPUブロック部は、図2に示されるマイクロコンピュータのレイアウトにおけるCPU200の部分を目指すものとする。尚、図2にお

いてFPUは浮動小数点演算ユニット、CCNはキャッシュ・コントローラ、BSCはバスステートコントローラ、TLBはトランスレーション・ルックアサイド・バッファを示し、これらのレイアウトは周知の構成である。

【0022】

同図1に示されるSIMDプロセッサは、64bitRISC (Reduced Instruction Set Computer) アーキテクチャを例として採用して、オペランドを3個有し、32bit固定長命令を実行するものとする。

【0023】

この図1の例では、演算パイプとして、データ整列(ALN)・乗算(MUL)・整数演算(INT)・ロード／ストア(LD／ST)・分岐(BRA)の5つをもつようなSIMDプロセッサを想定している。

【0024】

図3には、上記SIMDプロセッサのCPUブロック図のうち、命令デコード部からレジスタファイル213へのアクセス部について詳細に示してある。

【0025】

命令コード201のレジスタ指定フィールドの1つであるmには、R0～R63までのいずれかのレジスタ番号が指定され、そのコードはデコーダ202によりデコードされて、各レジスタに直接アクセスできる。

【0026】

このデコーダ202は、図4に示す真理値表を満たす論理回路で構成される。図4からもわかるように、出力64本のうち、4本がhighとなり、4つの連続したレジスタを1度に指定することができる。各バンクから読み出されたデータは、セクタ204とセクタ205へ出力される。セクタ205には、制御信号として、オペランドコードmの下位2bitを入力し、入力用の汎用バス206へ出力するデータを決定する。

【0027】

セクタ204にも同様に各バンクからのデータが入力され、汎用バスへ出力されるデータを除いた残りのデータを出力する制御信号としては、セクタ205と同じオペランドコードmの下位2bitを使う。

【0028】

図5にセクタ204と205の出力の様子を示す真理値表を示す。表中のaとbは、前述のオペランドコードmの下位2bitを意味し、セクタ205の出力が「To 206」と対応し、セクタ204の出力が「To 307」と対応する。又、この真理値表の「To 206」欄及び「To 307」欄の出力値として表現されているX0～X3の「X」はBNK（すなわち、バンク）を表している。

【0029】

これらセクタ204と205の出力は、各バンク毎にお互い排他的であり、この図5に示される真理値表を満たすような論理回路でセクタ204と205は構成される。

【0030】

各バンクについては、バンク3（図3内203に相当する）の詳細図が図6に示されているが、バンク3（203）に存在する各レジスタは、この例においては1bitあたり2read・1writeの標準的なもので構成される。

【0031】

図3において、バンク0には、R0・R4・…・R4nの16個のレジスタが格納され、バンク1には、R1・R5・…・R4n+1の16個、バンク2には、R2・R6・…・R4n+2の16個、バンク3には、R3・R7・…・R4n+3の16個が格納されている。

【0032】

図1に示す通り、各演算パイプで実行された演算結果は、出力用の汎用バス207に出力される。またこの例では、データ整列演算パイプや乗算パイプから出力が4本あるので、そのうち1本を汎用の207のバスへ、残りの3本を210のバスへ出力するようにする。210のバス上のデータは、セクタ208に入力される。

【0033】

セクタ208は3入力4出力のセクタで、図7に示される真理値表を満たす論理回路で構成される。

【0034】

図7中のaとbは、制御信号を示し、inputのX, Y, Zは入力値、outputは、左からバンク0、1、2、3への出力を示す。

【0035】

セクタ208に入力される制御信号は、ディスティネーションレジスタ指定フィールド（オペランドコードd）で示されるコードの下位2bitを使用する。また、汎用バス207へ出力されたデータは、セクタ209へ入力され、格納するレジスタが存在するバンクへ選択・出力される。

【0036】

セクタ209は、制御信号として、セクタ208同様ディスティネーションレジスタ指定フィールドで示されるコード（オペランドコードd）の下位2bitを使い、図8に示す真理値表を満たす論理回路で構成される。

【0037】

図8中のaとbは、ディスティネーションレジスタ指定フィールド（オペランドコードd）の下位2bitの値で、Xは入力値、Outputは、左からバンク0、1、2、3への出力を示す。

【0038】

これらセクタ208とセクタ209からの出力は、各バンクごとにお互いに排他的であり、片方のセクタの出力がデータならば、もう片方のセクタからの出力は0になる。従って、これら、セクタ208と209からの出力は、各バンクごとに論理和をとって、バンク内のレジスタへ書き込まれることになる。

【0039】

以上までが、複数レジスタが指定できるSIMDプロセッサの説明である。

【0040】

次に、このSIMDプロセッサの特徴を活かして定義されるSIMD演算命令について説明する。

【0041】

一般にSIMDプロセッサでは、レジスタ内のデータが即演算できる順番に整

列されているときに、その並列性を最大限に発揮することができるが、多くの場合データを並び替えて後に主要な演算を行う必要がある。従って、その並び替えサイクルをできる限り減らすことが、SIMDプロセッサの性能向上につながる。

【0042】

本発明の特徴を活かして、図9、図10のようなデータパック命令を定義する。

【0043】

図9はシフト量がレジスタ内に存在する場合。図10は即値として、命令コード内にシフト量が存在する場合である。

【0044】

図9ではデータパック命令のシフト量(R_n)をレジスタから読み込み、 R_m で示されるレジスタ群内のデータに対してシフト処理を施した後、パック演算を遂行するオペレーションコードを示しており、図10では図9と比較してシフト量(s)が即値の場合のオペレーションコードを示している。

【0045】

この命令は図9中の動作説明からも分かるように、4つのレジスタ内データを1つのデータにパックして格納するためのものである。

【0046】

この命令を実現するための回路構成は、図11に示されるものである。

【0047】

汎用バス206と307のバスには、レジスタ指定フィールド1（オペランドコード m ）で示される4個のレジスタ内データが転送される。またレジスタ指定フィールド2（オペランドコード n ）で示されるシフト量は、汎用バス301から取り込まれる。これらデータとシフト量は、任意 $b i t$ のシフトが可能なバレルシフタ302にそれぞれ入力される。このバレルシフタ302により固定小数点の除算を行うことができる。

【0048】

これらシフトされた結果の下位16 $b i t$ のみを各々取り出し、汎用バス20

7へ出力して、64bit幅データとして1つのレジスタにパックすることができる。

【0049】

次に本発明の特徴を活かして、図12、図13に示されるようなデータアンパック命令を定義する。このデータアンパック命令は、1つのレジスタ内SIMDデータを複数のレジスタに分割して格納する命令である。図12は16bitデータを扱う場合、図13は8bitデータを扱う場合を考慮して定義した。

【0050】

図12では64bitのデータを16bit毎に分割、分割されたデータを64bitに符号拡張し、ライトレジスタに書き込むオペレーションコードを示している。図13では図12と比較し、8bitデータを扱う為、書き込むライトレジスタ数が倍に成っている。

【0051】

この命令を実現するための回路構成を図14を用いて詳細に説明する。

【0052】

レジスタ指定フィールド1（オペランドコードm）で示されるレジスタ内のSIMDデータが汎用バス206を通して伝送される。

【0053】

この図14では、16bitデータに分割する場合を示してあるが、汎用バス206を通して送られたデータは、16bitごとに分割されて、それぞれ別の符号拡張器400に入力されて、64bitデータに符号拡張されてバス207と210に出力される。以上の操作でデータアンパック機能を実現することができる。

【0054】

さらに、本発明の特徴を活かして図15のような並べ替え命令を定義する。図15では、4つの指定されたレジスタ内SIMDデータを読み出して、要素毎に指定された並べ替えを行って指定されたレジスタに並べ替えたSIMDデータを格納する為のオペレーションコードを示している。

【0055】

一般的に「並べ替え命令」は、行列の転置や回転、FFTのバタフライ演算等に有効であり、具体的な動作は図16に示す。

【0056】

従来の並べ替え命令では、例えば16bit×4個のSIMDデータ2個に対して並べ替え操作を行うと、結果格納用に64bit幅レジスタ2個を必要とするが、ディスティネーションレジスタとして1個しか指定できないため、並べ替え結果の上位bit部分用と下位bit部分用にそれぞれ別命令を用意していた。

【0057】

本定義による並び替え命令は、1個のレジスタ指定フィールドで複数レジスタを指定できる特徴から、図16のように4個のソースデータに対して、一度に上位bit部と下位bit部の並べ替え操作を行うことができ、2組分を同時に演算することができる。

【0058】

図17に具体的な機能構成図を示す。汎用バス206と307のバスを通して伝送されるソースデータ対2組は、それぞれ16bit幅に分割されて、並び替え操作が行われ、その結果を207と210のバスへ出力して、レジスタに書き込まれる。

【0059】

また、複素数データのような場合、ロードしたデータには実数と虚数が交互に存在することが多く、演算には実数データのみ、虚数データのみのデータ列を必要とすることが多々ある。

【0060】

そのような場合、本発明のSIMDプロセッサならば、本実施例においては、最大で8個のSIMDデータを同時に読み込むことができるため、16bitデータならば32個のデータ間で並べ替え操作を行って、1度に16データ分の結果を求めることができる。

【0061】

上記のように複素数データを扱う処理を行うためには、図18、図19で示さ

れるような命令を定義すれば良い。

【0062】

図18では、8個の指定されたレジスタ内SIMDデータを読み出して、各要素を右端から数えて1・2・3・4としたとき、1と3のデータ要素のみを抽出して、指定されたレジスタに格納する処理を示すオペレーションコードを示しており、図19ではSIMDデータを読み出した後、2と4のデータ要素のみを抽出して、指定されたレジスタに格納する処理を示すオペレーションコードを示している。

【0063】

具体的には図20に示すような処理であって、この場合は各レジスタ内データの63bit目～48bit目のデータと、31bit目～16bit目のデータの抽出である。

【0064】

動作の詳細を図21の回路構成図で説明すると、汎用バス206と301、307と501のバスを通して伝送されるSIMDデータ8組32個は、それぞれ必要な16bitデータのみ抽出されて、結果を汎用バス207と210のバスへ出力することによって実現する。

【0065】

最後に、本発明の特徴を活かして、図22で示されるような積和演算命令を定義する。

【0066】

図22では、2個の指定されたレジスタ内SIMDソースデータを読み出し、4個の累積和の基になるレジスタ内データをさらに読み出し、累積和の計算をした後、指定された4個のレジスタにSIMDデータを格納する処理を示すオペレーションコードを示している。

【0067】

一般の乗算においては、乗数および被乗数のbit幅に対して、得られる結果は倍のbit幅をもつため、16bit幅のデータを4個も保持しているSIMD型64bitデータでは、演算結果を格納するために128bit幅のレジス

タが必要になる。現実解としては、SIMDの並列性を犠牲にして、64 bit 幅レジスタの下位32 bitにのみ有効なデータを格納して、結果を64 bit に収める方法が多くとられている。しかし累積和を取る場合、さらに結果の bit 幅が増加する可能性があり、上記方法でも演算精度が落ちてしまう。

【0068】

DSPにおいては、16 bit \times 16 bit の積和演算にて、格納用に40 bit レジスタを用意するなどして演算の精度を保つ工夫がなされている。

【0069】

しかし本発明の特徴を活かすと、SIMDの並列性を損なうことなく、精度の落ちない積和演算を実行することができる。

【0070】

図22で定義した積和演算命令の具体的な説明図を図23に、回路構成図を図24に示す。

【0071】

16 bit データ4個を含むSIMDデータは、汎用バス206とバス307を通して伝送される。それらのSIMDデータは、16 bit データごとに分割されて、各々乗算器700～703に入力され、704～707の加算器へ出力される。累積和のデータは、汎用バス301とバス501を通して伝送され、704～707の加算器へ入力される。これらの累積和演算の結果は、格納用の汎用バス207とバス210へ出力される。このような手段を用いて、精度を落とすことなく、かつ並列性を保ったまま、SIMDデータの積和演算を実現することができる。

【0072】

以上本発明者によってなされた発明を実施形態に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることは言うまでもない。

【0073】

例えば図1におけるセレクタ204は、4入力に対して3出力のセレクタであるが、これをトリステートバッファにしても何ら問題はない。

【 0 0 7 4 】

データパック命令の場合、図 1 1 では、4 入力 1 出力であるが、入力は何本でもよく、また出力本数も制限されるものではない。

【 0 0 7 5 】

さらにデータアンパック命令の場合、レジスタ内の SIMD データの型によって、分割する数が決定するので、図 1 4 のように 4 個に限定されるものでもない。

【 0 0 7 6 】

【発明の効果】

本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば下記の通りである。

【 0 0 7 7 】

すなわち、本発明による SIMD プロセッサは、SIMD 演算の効果を妨げるレジスタ内データ整列演算の高速化を実現することができ、さらに DSP 的な積和演算を実現することが可能である。

【 0 0 7 8 】

具体的な効果は以下に挙げる点である。

【 0 0 7 9 】

(1) 本発明により定義されたデータパック命令を導入することにより、ばらばらに格納されているデータを纏め、SIMD 命令で効率的にデータ処理を行うことができる。

【 0 0 8 0 】

本実施例の場合 (16 bit データ 4 個のパック) を図 2 5 に示す。図中の A は現状の場合のプログラム例であり、B が新規のデータパック命令を採用した場合のものである。

【 0 0 8 1 】

B 中の「pack. w」をデータパック命令のニーモニックとする。この A と B のプログラム例のように、データパック命令を採用することによって、命令ステップを 1 / 4 に削減することができる。

【 0 0 8 2 】

(2) 本発明により定義されたデータアンパック命令を導入することにより、速やかなレジスタの初期化や、データの分割が行え、S I M D 処理が効果的でない部分においても、演算効率が落ちないようにすることができる。

【 0 0 8 3 】

本実施例の場合（1 個のレジスタに存在する 1 6 b i t データ 4 個を 4 個のレジスタにアンパック）を図 2 6 に示す。

【 0 0 8 4 】

図中の A が従来の並べ替え命令を用いてデータパックする場合のプログラム例であり、B が新規データアンパック命令を採用した場合のものである。図 2 6 からわかるように、新規命令の追加によって、命令ステップ数を 1 / 6 にまで削減することができる。

【 0 0 8 5 】

(3) 本発明により定義されたデータ並べ替え命令を導入することにより、レジスタ内 S I M D データの並び替えを高速に行うことができ、積和演算等の S I M D 処理に遅滞なくデータを供給することができる。

【 0 0 8 6 】

本実施例の場合を図 2 7 に示す。図中の A が従来の並べ替え命令を用いた場合のプログラム例であり、図中の B が新規の並べ替え命令を採用した場合のプログラム例である。この図 2 7 から明らかなように、命令ステップ数を 1 / 8 にまで削減することが可能となる。

【図面の簡単な説明】

【図 1】

本発明の 1 例に係わる S I M D プロセッサの C P U ブロック図である。

【図 2】

本発明の 1 例に係わる S I M D プロセッサの全体ブロック図である。

【図 3】

前記 S I M D プロセッサのレジスタファイル部近辺の詳細図である。

【図 4】

デコード部を構成する論理回路の真理値表を示す図である。

【図 5】

図 3 のバンクからのデータを選択するセレクタを構成する論理回路の真理値表を示す図である。

【図 6】

バンク 2 0 3 内の個々のレジスタ詳細図である。

【図 7】

汎用バス上の演算結果を、どのバンクのレジスタに格納するのか、その場所をセレクトするためのセレクタを構成する論理回路の真理値表を示す図である。

【図 8】

バス 2 1 0 上の演算結果を、どのバンクのレジスタに格納するのか、その場所をセレクトするためのセレクタを構成する論理回路の真理値表を示す図である。

【図 9】

データパック命令の定義を示す図である。

【図 1 0】

即値を含むデータパック命令の定義を示す図である。

【図 1 1】

データパック命令を実現するための機能構成図である。

【図 1 2】

データアンパック命令の定義を示す図である。

【図 1 3】

8 b i t データに対するアンパック命令の定義を示す図である。

【図 1 4】

データアンパック命令を実現するための構成図である。

【図 1 5】

並べ替え命令の定義を示す図である。

【図 1 6】

並べ替え命令の具体的な説明図である。

【図 1 7】

並べ替え命令を実現するための機能構成図である。

【図 1 8】

並べ替え命令の定義を示す図である。

【図 1 9】

並べ替え命令の他の定義を示す図である。

【図 2 0】

並べ替え命令の具体的な使用例を示す図である。

【図 2 1】

図 2 0 を実現するための機能構成図である。

【図 2 2】

積和演算命令の定義を示す図である。

【図 2 3】

積和演算命令の具体的な説明図である。

【図 2 4】

図 2 3 を実現するための機能構成図である。

【図 2 5】

データパック命令導入の効果を示すプログラム例を示す図である。

【図 2 6】

データアンパック命令導入の効果を示すプログラム例を示す図である。

【図 2 7】

並べ替え命令導入の効果を示すプログラム例を示す図である。

【符号の説明】

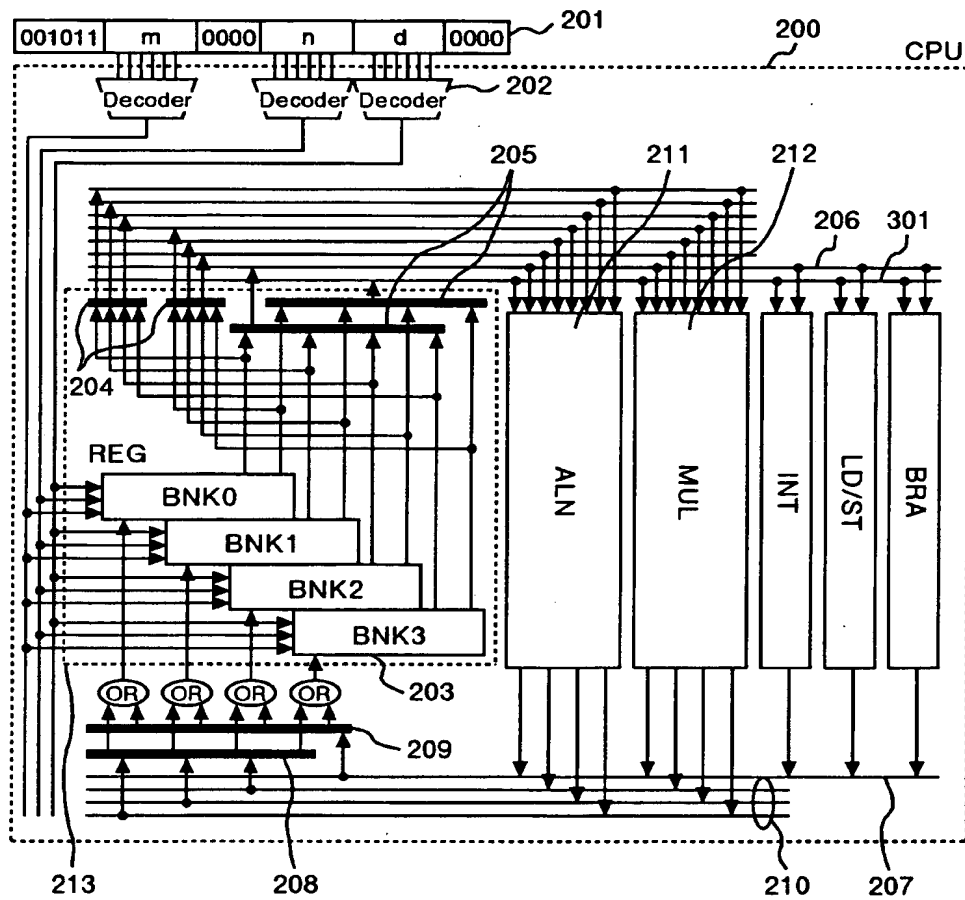
2 0 0 … CPU ブロック、2 0 1 … 命令コード、2 0 2 … レジスタ指定デコーダ、2 0 3 … レジスタファイルを分割したバンクの 1 つ、2 0 4 … 4 入力 3 出力セレクタ、2 0 5 … 4 入力 1 出力セレクタ、2 0 6、3 0 1 … ソースデータ用汎用バス、2 0 7 … 演算結果用汎用バス、2 0 8 … 3 入力 4 出力セレクタ、2 0 9 … 1 入力 4 出力セレクタ、2 1 0 … 演算結果用バス、2 1 1 … データ整列演算パイプ、2 1 2 … 乗算パイプ、2 1 3 … レジスタファイル、3 0 2 … 右算術シフト用バレルシフタ、3 0 7、5 0 1 … ソースデータ用バス、4 0 0 … 6 4 b i t 符号

拡張器、700、701、702、703…16bit乗算器、704、705
、706、707…64bit加算器、708、709、710、711…1b
it右シフト器。

【書類名】 図面

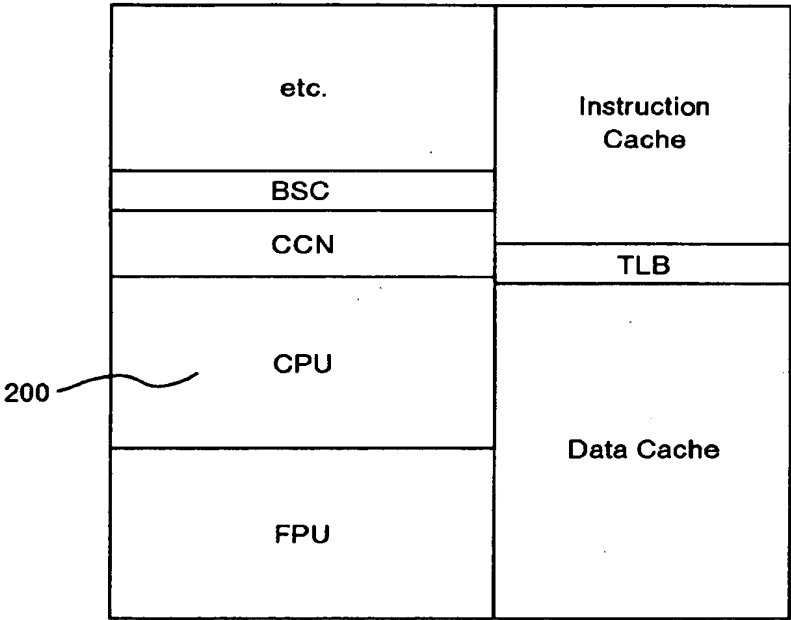
【図 1】

図 1



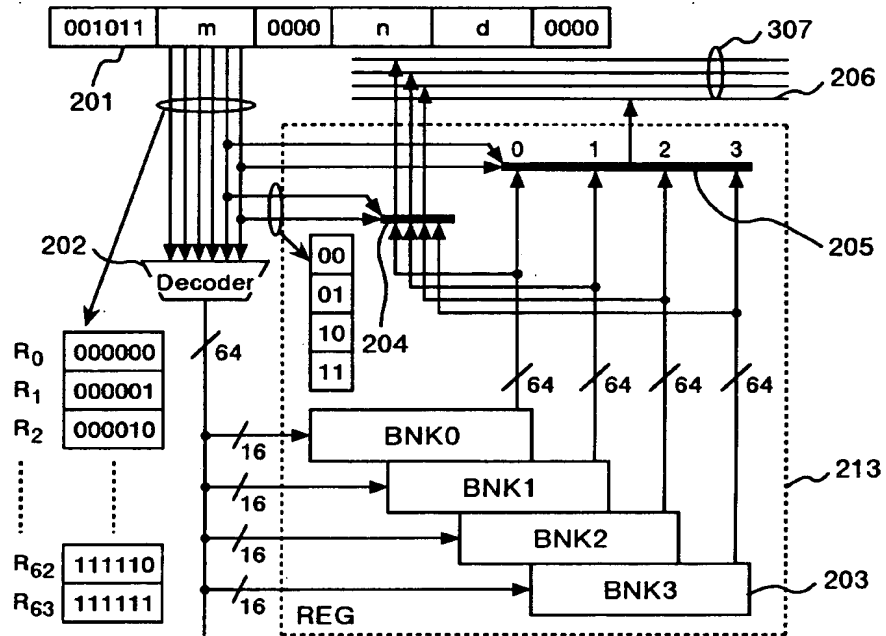
【図 2】

図 2



【図 3】

図 3



【図 4】

図 4

a	b	c	d	e	f	R0	R1	R2	R3	R4	R5	R6	R7	R8	R59	R60	R61	R62	R63
0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	1	1	1	0		0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	1	1	1	1		0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	1	1	1		0	0	0	0
		⋮								⋮							⋮		
1	1	1	0	1	1	0	0	0	0	0	0	0	0	0		1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0		0	1	1	1
1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0		0	0	0	1
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0		0	0	0	1

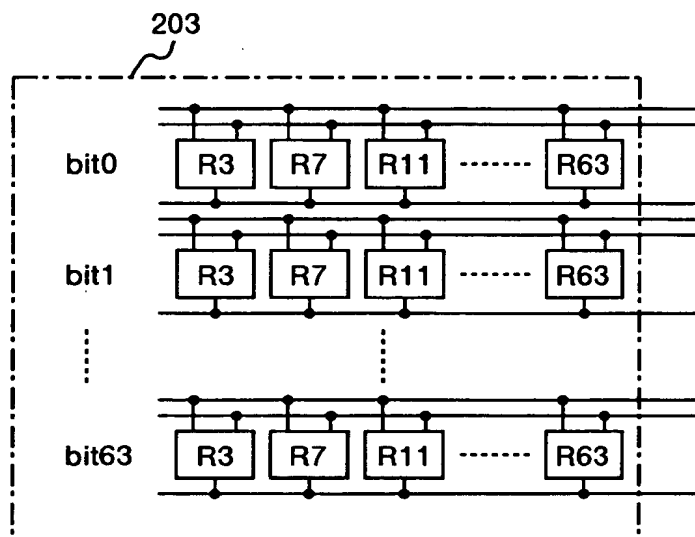
【図 5】

図 5

a	b	To 206	To 307		
0	0	X0	X1	X2	X3
0	1	X1	X2	X3	X0
1	0	X2	X3	X0	X1
1	1	X3	X0	X1	X2

【図 6】

図 6



【図 7】

図 7

a	b	input			output			
0	0	X	Y	Z	0	X	Y	Z
0	1	X	Y	Z	Z	0	X	Y
1	0	X	Y	Z	Y	Z	0	X
1	1	X	Y	Z	X	Y	Z	0

【図 8】

図 8

a	b	X	output			
0	0	X	X	0	0	0
0	1	X	0	X	0	0
1	0	X	0	0	X	0
1	1	X	0	0	0	X

【図 9】

図 9

PACK.W Rm, Rn, Rd

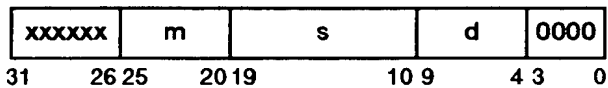
xxxxxx	m	xxxxx	n	d	0000
31	26 25	20 19 16 15	10 9	4 3	0

operation source [1] ← SignExtend₆₄(Rm);
 source [2] ← SignExtend₆₄(Rm+1);
 source [3] ← SignExtend₆₄(Rm+2);
 source [4] ← SignExtend₆₄(Rm+3);
 amount ← ZeroExtend₆₄(Rn);
 REPEAT i FROM 0 FOR 4
 result [i] ← ZeroExtend₁₆(source[i] >> amount);
 Rd ← MultiRegister₁₆(result);

【図 1 0】

図 1 0

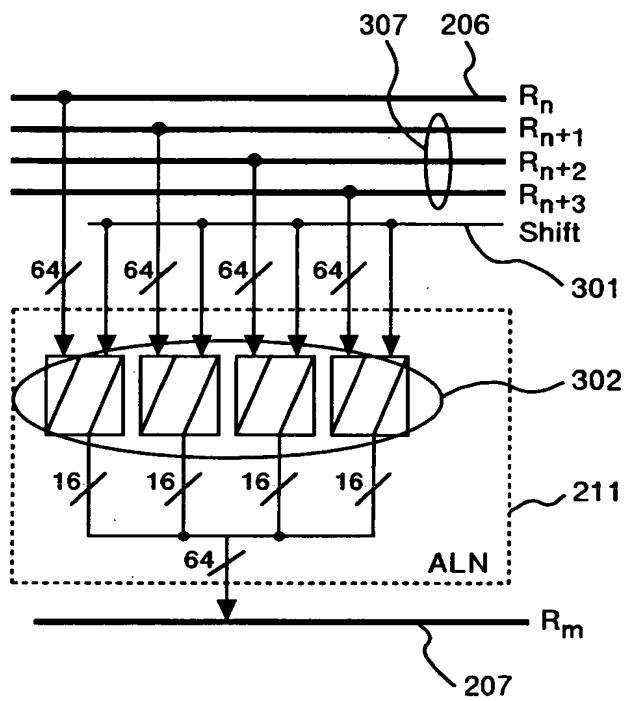
PACKI.W Rm, s, Rd



operation source [1] \leftarrow SignExtend₆₄(Rm);
 source [2] \leftarrow SignExtend₆₄(Rm+1);
 source [3] \leftarrow SignExtend₆₄(Rm+2);
 source [4] \leftarrow SignExtend₆₄(Rm+3);
 amount \leftarrow SignExtend₁₀(s);
 REPEAT i FROM 0 FOR 4
 result [i] \leftarrow ZeroExtend₁₆(source[i] >> amount);
 Rd \leftarrow MultiRegister₁₆(result);

【図 1 1】

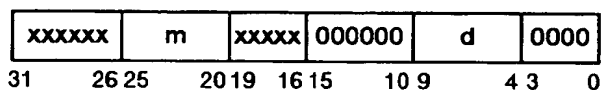
図 1 1



【図 1 2】

図 1 2

UNPACK.W Rm, Rd

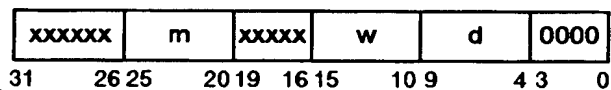


operation source \leftarrow SignExtend₆₄(Rm);
 REPEAT i FROM 0 FOR 4
 result[i] \leftarrow SignExtend₆₄(SignExtend₁₆(source[i]));
 Rd \leftarrow Register(result[0]);
 Rd+1 \leftarrow Register(result[1]);
 Rd+2 \leftarrow Register(result[2]);
 Rd+3 \leftarrow Register(result[3]);

【図 1 3】

図 1 3

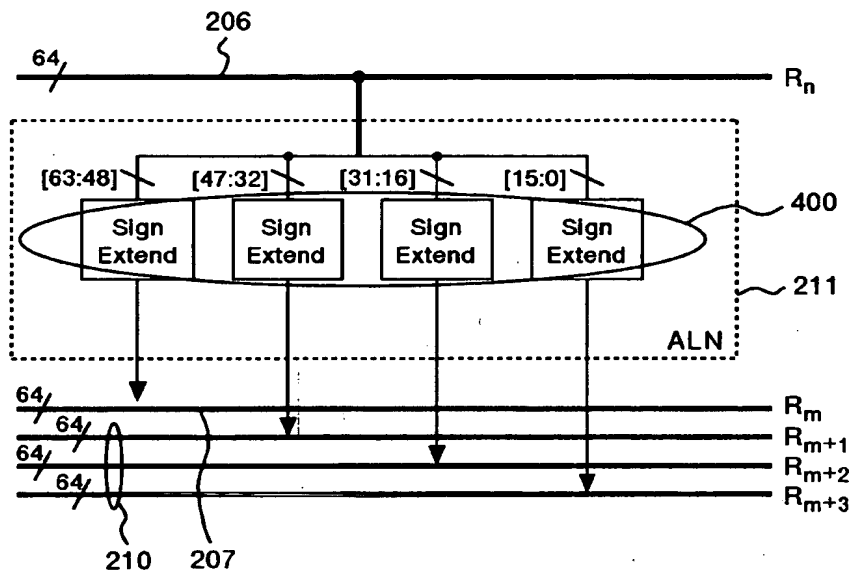
UNPACK.B Rm, Rw, Rd



operation source \leftarrow SignExtend₆₄(Rm);
 REPEAT i FROM 0 FOR 8
 result[i] \leftarrow SignExtend₆₄(SignExtend₈(source[i]));
 Rw \leftarrow Register(result[0]);
 Rw+1 \leftarrow Register(result[1]);
 Rw+2 \leftarrow Register(result[2]);
 Rw+3 \leftarrow Register(result[3]);
 Rd \leftarrow Register(result[4]);
 Rd+1 \leftarrow Register(result[5]);
 Rd+2 \leftarrow Register(result[6]);
 Rd+3 \leftarrow Register(result[7]);

【図 1 4】

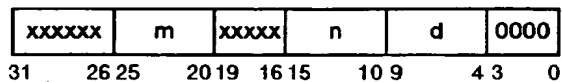
図 1 4



【図 1 5】

図 1 5

MSHFLE.W Rm, Rn, Rd



operation

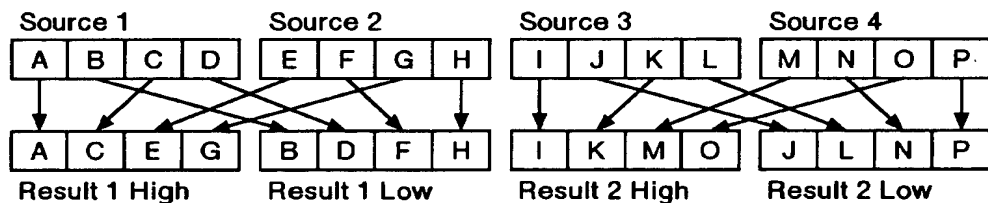
```

source1 ← MultiZeroExtend16(Rm);
source2 ← MultiZeroExtend16(Rm+1);
source3 ← MultiZeroExtend16(Rn);
source4 ← MultiZeroExtend16(Rn+1);
REPEAT i FROM 0 FOR 2
{
result1[i x2] ← source2[i+2]; result1[(i x2)+1] ← source1[i+2];
result2[i x2] ← source2[i];   result2[(i x2)+1] ← source1[i];
result3[i x2] ← source4[i+2]; result3[(i x2)+1] ← source3[i+2];
result4[i x2] ← source4[i];   result4[(i x2)+1] ← source3[i];
}
Rd ← MultiRegister16(result1);
Rd+1 ← MultiRegister16(result2);
Rd+2 ← MultiRegister16(result3);
Rd+3 ← MultiRegister16(result4);

```

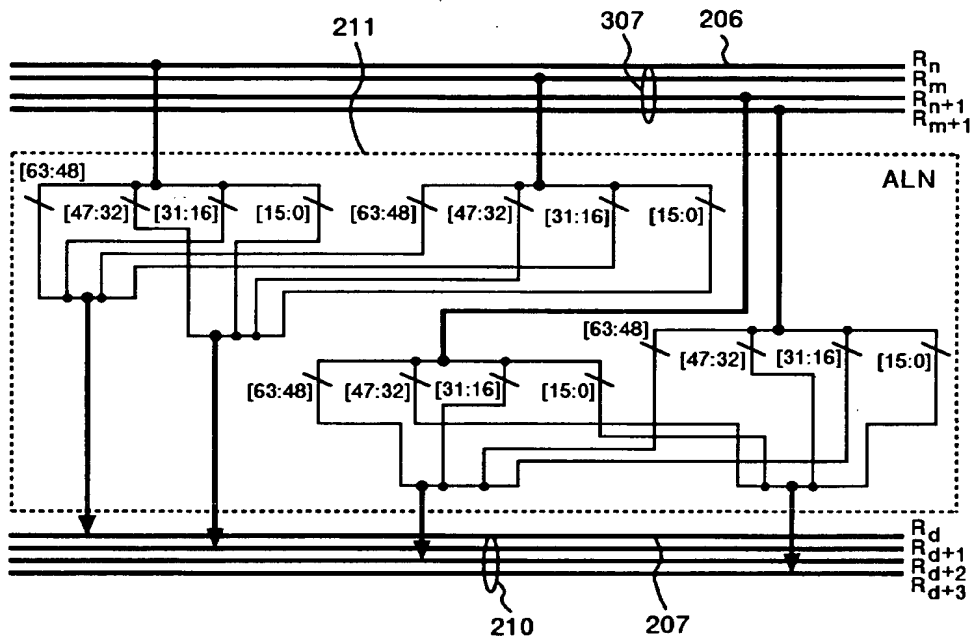
【図 1 6】

図 1 6



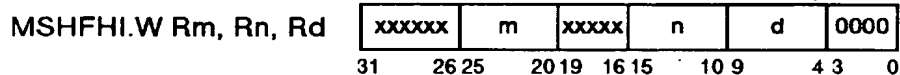
【図 1 7】

図 1 7



【図 1 8】

図 1 8

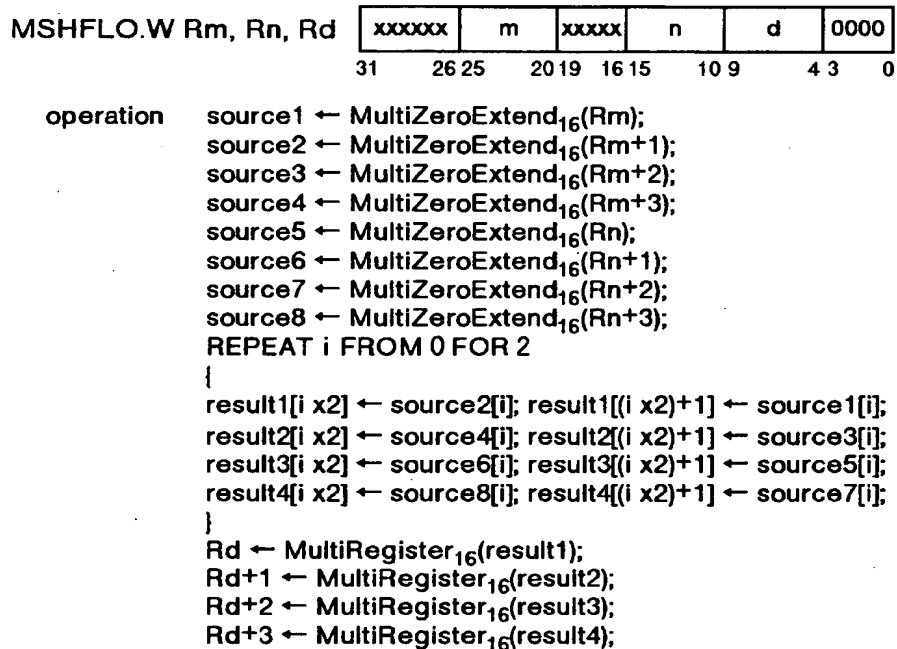


```

operation  source1 ← MultiZeroExtend16(Rm);
            source2 ← MultiZeroExtend16(Rm+1);
            source3 ← MultiZeroExtend16(Rm+2);
            source4 ← MultiZeroExtend16(Rm+3);
            source5 ← MultiZeroExtend16(Rn);
            source6 ← MultiZeroExtend16(Rn+1);
            source7 ← MultiZeroExtend16(Rn+2);
            source8 ← MultiZeroExtend16(Rn+3);
            REPEAT i FROM 0 FOR 2
            {
            result1[i x2] ← source2[i+2]; result1[(i x2)+1] ← source1[i+2];
            result2[i x2] ← source4[i+2]; result2[(i x2)+1] ← source3[i+2];
            result3[i x2] ← source6[i+2]; result3[(i x2)+1] ← source5[i+2];
            result4[i x2] ← source8[i+2]; result4[(i x2)+1] ← source7[i+2];
            }
            Rd ← MultiRegister16(result1);
            Rd+1 ← MultiRegister16(result2);
            Rd+2 ← MultiRegister16(result3);
            Rd+3 ← MultiRegister16(result4);
    
```

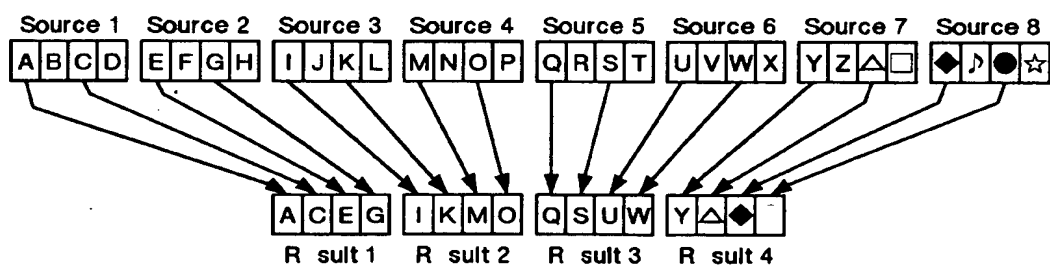
【図 1 9】

図 1 9



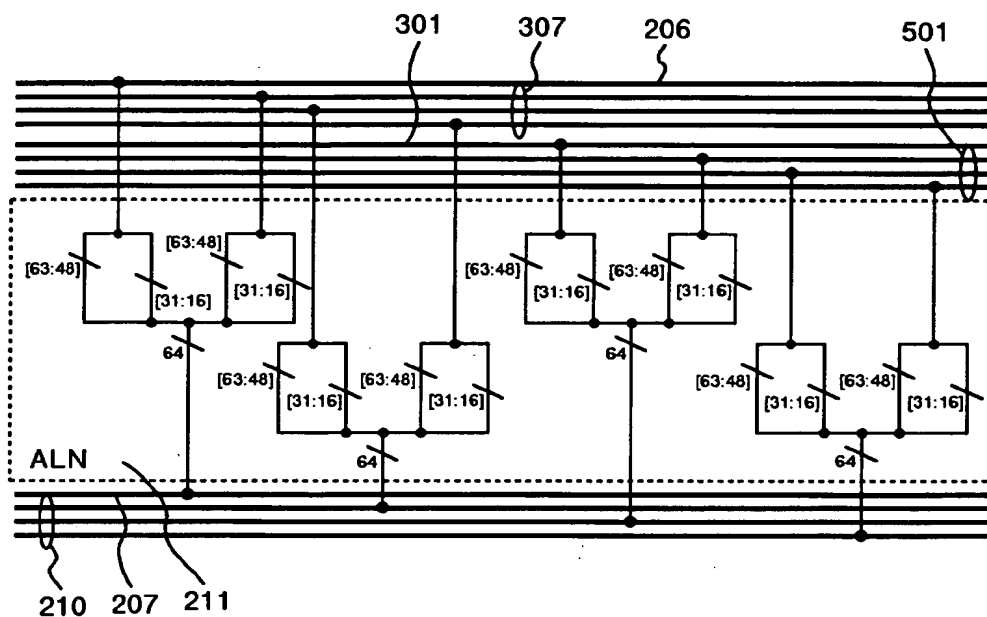
【図 2 0】

図 2 0



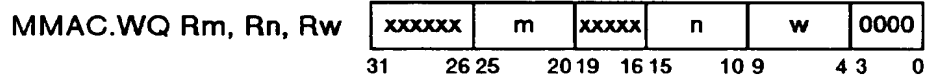
【図 2 1】

図 2 1



【図 2 2】

図 2 2



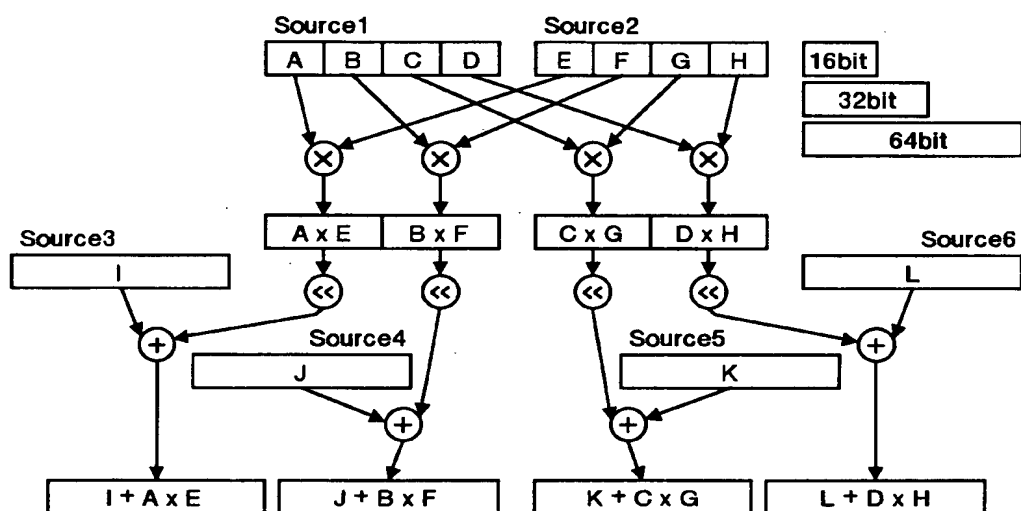
```

operation  source1 ← MultiSignExtend16(Rm);
           source2 ← MultiSignExtend16(Rm+1);
           result[0] ← SignExtend64(Rn);
           result[1] ← SignExtend64(Rn+1);
           result[2] ← SignExtend64(Rn+2);
           result[3] ← SignExtend64(Rn+3);
           REPEAT i FROM 0 FOR 4
           {
             temp ← source1[i] x source2[i];
             temp ← SignedSaturate64(temp << 1)
             result[i] ← SignedSaturate64(result[i] + temp)
           }
           Rw ← Register(result[0]);
           Rw+1 ← Register(result[1]);
           Rw+2 ← Register(result[2]);
           Rw+3 ← Register(result[3]);

```

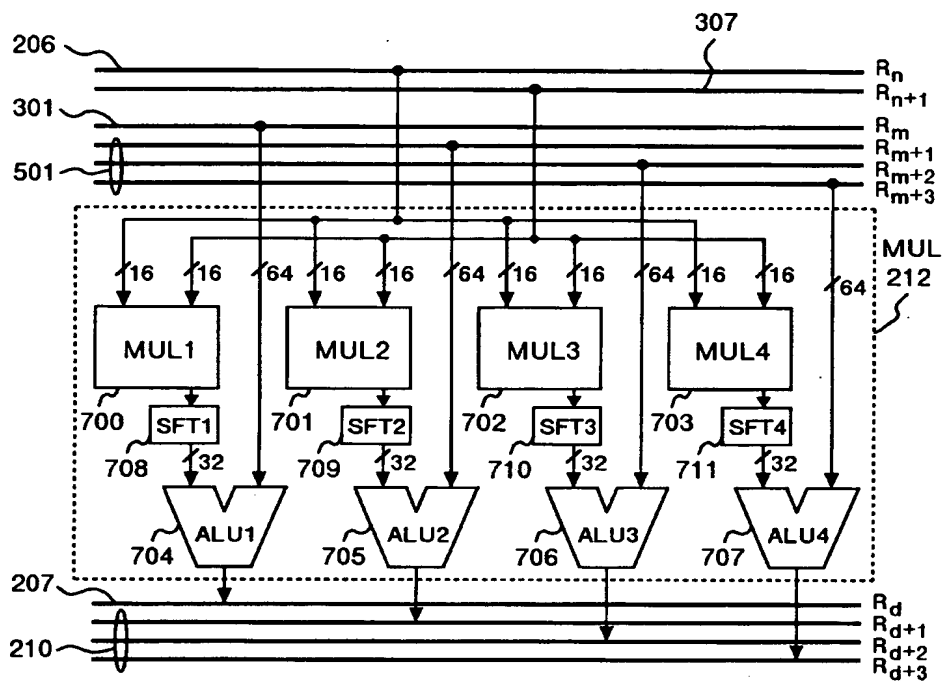
【図 2 3】

図 2 3



【図 2 4】

図 2 4



【図 2 5】

図 2 5

A		B	
mulu.l	r51, r24, r51	mulu.l	r51, r24, r51
mulu.l	r52, r24, r52	mulu.l	r52, r24, r52
mulu.l	r53, r24, r54	mulu.l	r53, r24, r54
mulu.l	r54, r24, r54	mulu.l	r54, r24, r54
shlri	r51, 9, r51	pack.w	r51, 9, r55
shlri	r52, 9, r52		
shlri	r53, 9, r53		
shlri	r54, 9, r54		
st.w	r17, 0, r51	st.q	r17, 0, r55
st.w	r17, 2, r52		
st.w	r17, 4, r53		
st.w	r17, 6, r54		

【図 2 6】

図 2 6

A		B	
mshfhi.w	r17, r63, r20	unpack	r17, r17
mshflo.w	r17, r63, r17		
mshfhi.l	r20, r63, r19		
mshflo.l	r20, r63, r20		
mshfhi.l	r17, r63, r21		
mshflo.l	r17, r63, r17		

【図 2 7】

図 2 7

A		B	
mshfhi.w	r21, r6, r31	mshfle.w	r21, r6, r41
mshflo.w	r21, r6, r21		
mshfhi.w	r31, r21, r41		
mshflo.w	r31, r21, r42		
mshfhi.w	r22, r7, r32		
mshflo.w	r22, r7, r22		
mshfhi.w	r32, r22, r43		
mshflo.w	r32, r22, r44		

【書類名】 要約書

【要約】

【課題】 SIMDプロセッサの高速化においては、レジスタ内データ整列など、SIMD演算の効果を妨げる要因を軽減する必要がある。

【解決手段】 レジスタファイルを4個のバンクに分けて、1個のオペランドで複数個のレジスタを指定できるようにして、4個のレジスタを同時にアクセスできるようにすることによって、データ整列演算パイプ211に多数のデータを供給でき、高速にデータ整列演算を行うことができる。また、新規のデータパック命令・データアンパック命令・データ並べ替え命令を定義することによって、その多量に供給されるデータを効率よく整列させることができる。さらに、上記の特徴により、SIMDの並列性を最大限に生かした積和演算命令の定義が可能である。

【選択図】 図1

出 願 人 履 歴 情 報

識別番号 [0 0 0 0 0 5 1 0 8]

1. 変更年月日	1 9 9 0 年 8 月 3 1 日
[変更理由]	新規登録
住 所	東京都千代田区神田駿河台4丁目6番地
氏 名	株式会社日立製作所